International Journal of Aquatic Science
ISSN: 2008-8019
Vol 12, Issue 02, 2021

Frequent Sub-Graph Mining Using Missing Items

Dr.B.Senthil kumaran

Assistant Professor & Head PG & Research Department of computer science Jairams Arts and Science College (affiliated to Bharathidasan University) karur.3

skumaran.gac16@gmail.com

ABSTRACT: One of the most sought after research area is graph mining and extracting the hidden patterns from the graphs is a tedious task and that too unearthing meaningful patterns is a challenging process and this paper focus on discovering useful patterns from the graph data using a new algorithm named "Frequent Subgraph miningusing missing items – FSMM algorithm" is employing some simple mechanisms to evade the consumption of excess runtime and memory allocation. The graphs are initially converted into textual transaction data where only the missing items are considered and this is transformed into binary representations to discover the frequent sub-graphs. The results are experimentally evaluated with state of the art existing algorithms to prove the performance of the proposed algorithm.

Keyword: Graph mining – FIM – subgraph mining – data mining – binary representations

1. INTRODUCTION

The pioneer in graph mining domain is frequent sub-graph mining (FSM) and this area is not new. The preeminent target of FSM is to find all frequent sub-graphs in a given graph dataset whose event is over the limit check esteem given by the user to discover the sub-graphs.

The fundamental procedure behind FSM is to create candidates (sub-graph candidates) using depth first or breadth first techniques and by employing the support consideration provided by the user [4].

To the extent the FSM is viewed as two significant issues must be dealt with proficiently

(I) Finding the entire candidate frequent sub-graphs without any redundancy.
(II) Eliminate the recurrence check of the created sub-graphs to reduce the time complexity.

Here care must be taken to stay away from the generation of copy or superfluous candidates. Backing tally checking requires redundant correlation of candidate sub-graphs with sub-graphs in the information data and FSM can be considered as an expansion of Frequent Itemset Mining (FIM) promoted with regards to ASM [4]. Numerous scientists proposed answers for address the issues identified with FSM and descending conclusion property related with itemset mining is generally taken on for candidate sub-graph age. This paper

manages many best in class FSM based algorithms utilizing various strategies regarding time complexity, memory complexity and search space related issues.

The frequent sub-graph mining is classified into two major categories, namely

1. Mining using collection of graphs and discover the frequent sub-graphs
2. Mining using single large graph and discover the frequent sub-graphs.

Let us consider a graph dataset Gd = { G_1, G_2, G_3,…. G_N} Where G1, G2, G3 are collection of various graphs given in the dataset, the minimum support count threshold σ (0 <σ≤ 1). Then the support of M is

$$MinSup(M) = | \delta(M) | / N$$

Where | δ(M) | is cardinality of δ(M) and N is total number of graph presents in the graph dataset. Here M is frequent, if Sup(M) ≥σ.The idea is simple if the superset is frequent, then all the subsets are also frequent.

2. PROPOSED WORK

The proposed algorithm work is based on single large dataset and the sample dataset is shown in the figure 1. Initially the graph is traversed from the top to bottom and the vertices, nodes and edges are found to convert the graphical data into textual transaction data. The converted transaction data is then represented with only the missing items in each row. The missing data representation is converted into binary presentation and then the probable candidates are formed using simple binary operation and user provided minimum support count threshold value.

The procedure to convert the graph data to text transaction is shown in the following figure 3. The initial process is to find the edges present in each and every node. Let us find the number of edges present in the first node N1.

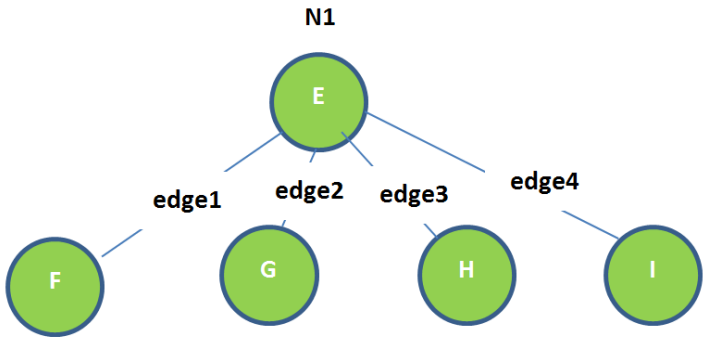

Figure 1: Single graph dataset

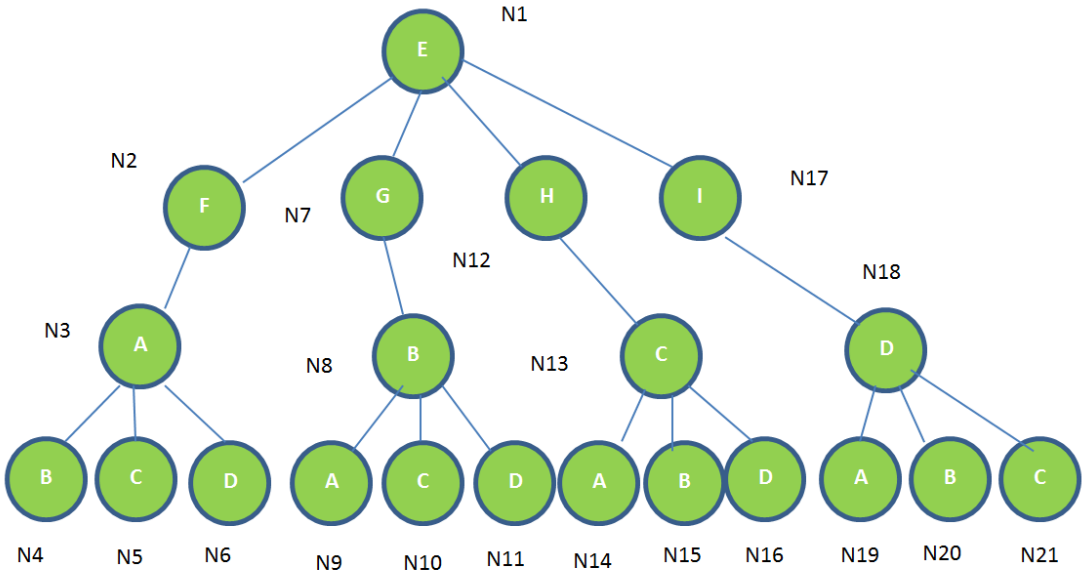

Figure 2: Finding the edges

PROCEDUREConvertGraphToText(Graph G)

Input: Single Graph data G
Output: Textual Labels with number of edges present in them

1. Load the single Graph dataset
2. Initially Scan the graph dataset to find the number of levels
3. Detect the vertexes present in the graph
4. Initialize edge =1 // to find the number of edges
5. ∀ Vertex V in Graph G
   Discover the edges connected with the v
   Fetch the labels of the node
   Increment edge = edge + 1
   Store the labels and edge → RES
6. End For
7. Return RES

Figure 3: Pseudo code to convert graph to text

Here the number of edges is found to be 4 as shown in the figure 2. Similarly for all nodes the edges are found.

NODE	ITEMS	EDGES	NODE	ITEMS	EDGES
N1	**E, F, G, H, I**	4	N11	**D, B**	1
N2	**F, E, A**	2	N12	**H, E, C**	2
N3	**A, F, B, C, D**	4	N13	**C, H, A, B, D**	4
N4	**B, A**	1	N14	**A, C**	1
N5	**C, A**	1	N15	**B, C**	1
N6	**D, A**	1	N16	**D, C**	1
N7	**G, C, B**	2	N17	**I, E, D**	2
N8	**B, G, A, C, D**	4	N18	**D, I, A, B, C**	4
N9	**A, B**	1	N19	**A, D**	1
N10	**C, B**	1	N20	**B, D**	1

Table 1: Converted graph to text

The graph that are converted into the textual data is shown in the table 1 and from this table the unique items are found and after scrutinizing the minimum support count the missing item representation is processed as shown in the following section. The procedure to find the distinct items is shown in the following figure 4 and the pseudo code is showcased,

PROCEDURE DistinctItems(Database Đs)

INPUT: Transactional Database Đs
OUTPUT: Distinct items with their Count

1. Load and Scan the database Đs
2. Initialize the Result[ ] = ϕ
3. ∀ Transactional Row Ř ∈ Đs do
4. ∀ Item I ∈ Row Ř do
5. IF (I not present in Result[ ]) then
6. Store Item I in Result[ ]
7. else
8. Increment Count Ci[] by 1
9. End IF`
10. End FOR
11. End FOR
12. Return Result[] with Count Ci[]

END PROCEDURE

Figure 4: Pseudo code to find the distinct items

The support count is provided by the user is 4 and the distinct items are found to be {A, B , C, D, E, F, G, H, I, } where the item count of F, G, H, I are found to be 3 and as it is lower than the user defined support, those items are pruned. The pruned transactional data representation is shown in the following table 2 and from this table the missing items are discovered.

Table 2: Pruned transactional data

NODE	ITEMS	NODE	ITEMS
N1	**E**	N11	**D, B**
N2	**E, A**	N12	**E, C**
N3	**A, B, C, D**	N13	**C, A, B, D**
N4	**B, A**	N14	**A, C**
N5	**C, A**	N15	**B, C**
N6	**D, A**	N16	**D, C**
N7	**C, B**	N17	**E, D**
N8	**B, A, C, D**	N18	**D, A, B, C**
N9	**A, B**	N19	**A, D**
N10	**C, B**	N20	**B, D**

Table 3: Missing item transactional data

NODE	ITEMS	NODE	ITEMS
N1	**A,B,C,D**	N11	**A,C,E**
N2	**B,C,D**	N12	**A,B,D**
N3	**E**	N13	**E**
N4	**C,D,E**	N14	**B,D,E**
N5	**B,D,E**	N15	**A,D,E**
N6	**B,C,E**	N16	**A,B,E**
N7	**A,D,E**	N17	**A,B,C**
N8	**E**	N18	**E**
N9	**C,D,E**	N19	**B,C,E**
N10	**A,D,E**	N20	**A, C,E**

Here the items that are missed in each node are found and represented as shown in the following table 3.

Table 4: Binary representation of the missing items

NODE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A	1	0	0	0	0	0	1	0	0	1	1	1	0	0	1	1	1	0	0	1
B	1	1	0	0	1	1	0	0	0	0	0	1	0	1	0	1	1	0	1	0
C	1	1	0	1	1	1	0	0	1	0	1	0	0	0	0	0	1	0	1	1
D	1	1	0	1	0	0	1	0	1	1	0	1	0	1	1	0	0	0	0	0
E	0	0	1	1	1	1	1	1	1	1	1	0	1	1	1	1		1	1	1

Figure 5: Pseudo code to represent the missing item dataset in binary format

PROCEDURE BinaryValue(Missing Item M)

Input: Missing Item M
Output: Bit vector representation of data

1. Load missing item Data set M and scan it.
2. ∀ Row Ro∈M begin
3. ∀ Item It ∈Ro begin
4. If [It present in Ro] begin
5. Mark as "1" in out
6. Else
7. Mark as "0" in out
8. Close IF
9. Close For
10. Close For
11. Return out

The table 3 is considered and if the item is present in the node then it is marked by one and else marked by zero. The pseudo code to perform this task is shown in the figure 5.

The frequent graphs are found by computing the probable candidate and lets us start from the 2 itemset value A and B.

A = 10000010011100111001
B = 11001100000101011010 |OR
AB = 11 0 0 1 1 1 0 0 1 1 1 0 1 1 1 1 0 1 1

Now count the number of zeroes and from the result {AB}=11001110011101111011 where the number of zero is 6 and the user defined support provided is 4. The count of AB is higher than the minimum support and this graph is found to be frequent. Now the next level 3-itemset is found using this {AB} | {C}

AB = 1 1 0 0 1 1 1 0 0 1 1 1 0 1 1 1 1 0 1 1
C = 1 1 0 1 1 1 0 0 1 0 1 0 0 0 0 0 1 0 1 1 | OR
ABC = 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1

The number of zeroes is found to be 3 and it is lesser than the minimum support count and hence it is infrequent and the next level 4-itemset is ignored. The final frequent sub-graph is shown in the following table 5.

Table 5: Final result

OUTPUT for minimum support →4			
Frequent Itemset	**Count**	**Frequent Itemset**	**Count**
AB	**6**	**AD**	**6**
ABC	**4**	**BC**	**6**
ABCD	**4**	**BCD**	**4**
AC	**6**	**BD**	**6**
ACD	**4**	**CD**	**6**

PROPOSED ALGORITHM

ALGORITHM FSMM

Input: Graph Database G , min_sup
Output: Frequent Items

1. Load the graph dataset
2. Convert graph to test →TextTable
3. Load the transaction table Texttable
4. Find the missing Value → Tm
5. Bin=BinaryValue(dataset Tm)
6. Find the level wise calculation
7. Count the number of zeroes
8. If [Count >= min_sup]
9. Store the itemset in RES
10. Calculate the next level itemset
11. Else
12. Prune the Itemset
13. Return RES

Figure 6: Pseudo code of the proposed algorithm FSMM

3. EXPERIMENTAL EVALUATION

The proposed FSMM algorithm is executed on a system comprising of 2.66 GHz I7 processor machine with a 4 GB memory running on Microsoft 10 ultimate operating system. The algorithm is written in java based SPMF data mining toolkit. The proposed algorithm was compared with the existing algorithms like FSG [1], GSPAN [2], GASTON [3] and the results are showcased to prove the effective working of the proposed algorithm.

Kuramochi and Karypis [1] proposed the FSG algorithms for mining all frequent sub-graphs from graph datasets, using a level-wise approach based on the Apriori concepts and this algorithm was the first one compared with the proposed FSMM.

The author Yan and Han [2] proposed GSPAN, which employed depth- first search, based on a pattern growth principle similar to the FP-growth algorithm and here the candidate generation is evaded but this requires lot of memory.

The author Nijssen et al. proposed a more efficient frequent sub-graph mining tool, called Gaston, which discovers the frequent substructures in a number of phases of increasing complexity [3].

The synthetic data generator first creates a set of candidate graphs (the total number is controlled by L) with user specified size (I). The parameters used in the synthetic graph generator are shown in the table 6.

Table 6: Parameter used in synthetic dataset

Parameters	**Description of parameter**
D	Total number of graph
L	Total number of probable frequent sub-graph present
T	Number of Edges
V	Label count
I	Edge size
E	Edge label count

The dataset generated is shown in the following table and three datasets are generated and employed for experimental comparison with respect to the runtime and memory consumption.

Table 2: Pruned transactional data

Dataset generated	**Number of Sequences**	**Avg. edges**	**Potential freq patterns**
D15kT30L200I11V4E4	10000	35	250
D25kT40L350I16V4E4	20000	45	375
D120kT50L500I20V4E4	100000	55	550

X

The proposed FSMM algorithm along with the other existing algorithms is executed on the synthetic datasets shown in the table 7. The results obtained after execution with respect to runtime consumption is noted as shown in the table 8.

RUNTIME (mSEC)	
Dataset - D15kT30L200I11V4E4	
Algorithm	**User defined Min_Sup values**

	10	**100**	**1000**	**2000**	**2500**
FSG	1244	947	831	787	659
GSPAN	1219	894	726	678	646
GASTON	1079	859	717	626	538
FSMM	978	722	665	556	437

Table 8: Run time comparison on D10kT30L200I11V4E4 (Small) dataset

The experimental results showcased in the table 8 clearly indicate that the proposed FSMM algorithm performs reasonably well small synthetic dataset. The GASTON algorithm almost matched the performance of the FSMM when a large support count is provided but the other two algorithms performed quite badly and took quite a lot of time to execute.

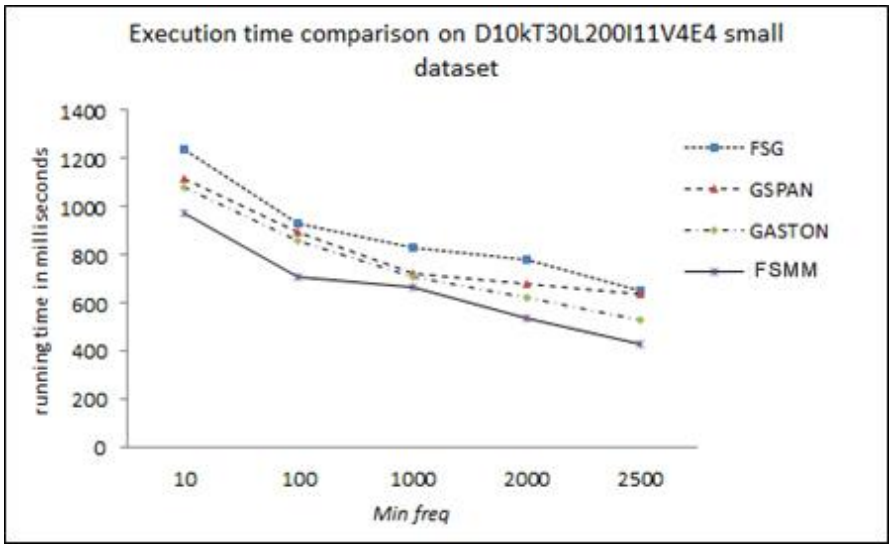

Figure 7: Graph related to runtime comparison

The next comparison is carried out with memory footprint and the following table 9 showcases the comparison of memory consumption.

MEMORY USAGE (MB)					
Dataset - D15kT30L200I11V4E4					
Algorithm	**User defined Min_Sup value**				
	10	**100**	**1000**	**2000**	**2500**
FSG	323	109	78	58	35
GSPAN	276	93	66`	42	29
GASTON	224	79	57	37	27
FSMM	168	66	45	29	21

Table 9: Memory consumption comparison

The table 9 clearly demonstrates that the proposed FSMM algorithm outscores the other three algorithms by a good margin with respect to the memory consumption. When the minimum

support value is reduced below 5, FSG suffered with out of memory error and when the minimum support value is increased above 3000 almost all the algorithms performed equally well.

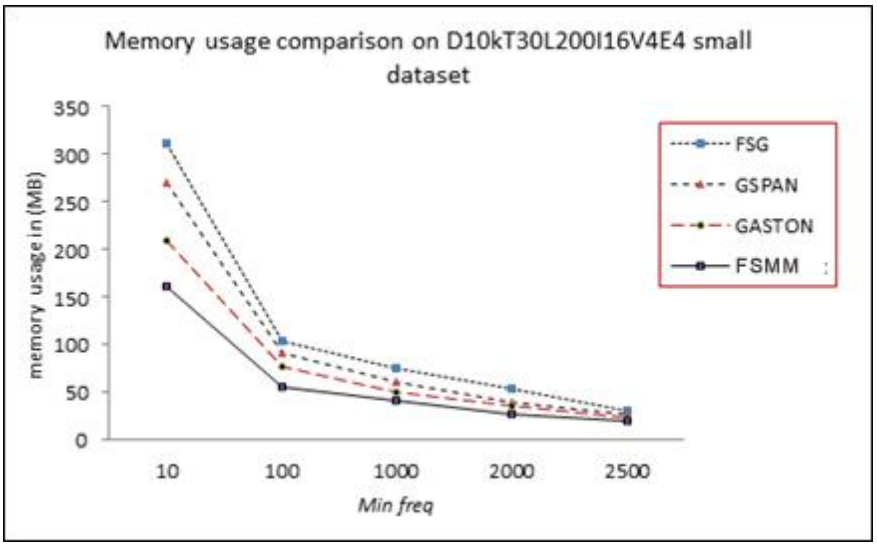

Figure 8: Graph related to memory comsumption

4. CONCLUSION

The proposed algorithm FSMM is showcased in this paper and from the experimental results it is quite clear that the FSMM outperformed the other algorithms by a large magnitude regarding runtime and memory consumption. The proposed method clearly saved lot of time and memory when executed on large graph dataset and proved to be an asset to the research communities.

5. REFERENCES

[1] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proc. International Conference on Data Mining'01*, 2001.
[2] Yan, X. and Han, J.W. 2002. gSpan: Graph-based Substructure pattern mining, In Proceedings of International Conference on Data Mining, 721–724.
[3] S. Nijssen and J.N. Kok. The gaston tool for frequent subgraph mining. Electronic Notes in Theoretical Computer Science, 127:77-87, 2005.
[4] Chen, M.S., Han,J.andYu,P.S. 1996 Data mining – An overview from database perspective, IEEE *Transaction on knowledge and data engineering* 8 , 866-883
[5] http://cygnus.uta.edu/subdue/databases/index.html
[6] http://citeseer.ist.psu.edu/oai.html
[7] http://vlsicad.cs.ucla.edu/cheese/ispd98.html